Degree in Mathematics

Title: ECG heartbeat classification using deep neural networks

Author: Gerard Riera Solà

Advisor: Jordi Fonollosa Magrinya

Department: Facultat de Matemàtiques i Estadística

Academic year: 2017-2018

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

ECG heartbeat classification using deep neural networks
Bachelor's thesis
Universitat Politècnica de Catalunya

Gerard Riera Solà
Advisor: Jordi Fonollosa Magrinya

September 2018

Abstract

The goal of the present work is to develop a method that performs automatic classification of beat types in ECG recordings, which has its motivation in the difficulty of physicians to manually classify certain types of beats, coupled with the large amounts of data avaliable for analysis.

We present a proposal for a machine-learning based solution and compare its performance with classical algorithms and current related research. Our approach performs automatic feature selection by means of a convolutional autoencoder, followed by classification by a dense softmax layer. No auxiliary software for feature extraction is used and minimal preprocessing is applied, allowing for greater automation of the process, as well as real-time usage. Results show a performance comparable to existing work, while having the aforementioned advantages. We achieved class sensitivities of 80% for class S, 69.3% for class N, 85.9% for class V and 58.1% for class F. We show that the obtained figures are comparable to those obtained in several current works, while achieving particularly good results in the classification of F-beats.

Keywords: Heartbeat classification, bioinformatics, machine learning, deep learning, autoencoders, convolutional neural networks

Contents

1 Introduction

According to the World Health Organization, ischaemic heart disease and stroke are the main causes of death in the world and have remained so for the past 15 years [1]. As such, effective tools for the diagnosis of heart-related pathologies are crucial.
The main tool for this purpose is the electrocardiogram (ECG), which is the recording of electrical activity of the heart over a period of time by using a number of electrodes placed in several locations on the pacient's skin. They measure the heart's electrical potential, capturing its polarizations and depolarizations during its periodic contractions, generating a graph of voltage over time.
Due to the sometimes low occurrence of some anomalous beats, long samples have to be taken and as such cardiologists have to invest a significant amount of time analyzing each recording. Moreover, in diseases like Brugada's Syndrome [3] anomalous beats can be hard to distinguish even by specialists.
In view of these difficulties, aggravated by the ever-growing amount of data that is at the physician's disposition, it is clear that the development of an effective means for automatic beat classification is of high utility.

1.1 Heartbeat anatomy

A hertbeat has a complex physical behaviour, however it has key distictive features to which we will refer throughout the work:
A normal heartbeat consists of five waves, described in cronological order: the P wave, the Q,R and S waves, which conform its central part and are referred to as the QRS complex, and the T wave.
Upon inspection of the ECG, features can be derived that help categorize each type of beat, like the duration of certain waves and offset between one wave and another (see Figure 1).
Additionally, the distance between one R-wave peak and the next or previous one can be calculated (R-R interval).
According to recommendations by the Association for the Advancement of Medical Instrumentation (AAMI) [4], heartbeats can be classificated into five broad classes: N, S, V, F and Q, whose composition is detailed

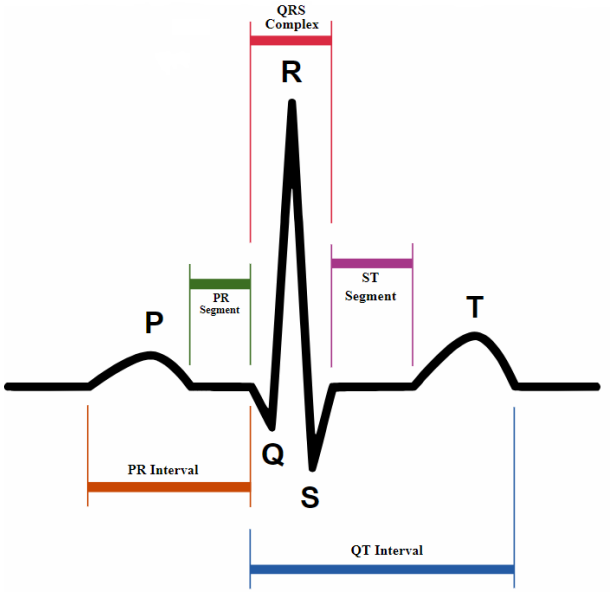


Figure 1: Main beat characteristics

in Figure 2.

heartbeat class	N	S	V	F	Q
Description	Any heartbeat not in the S, V, F or Q classes	Supraventricular ectopic beat	Ventricular ectopic beat	Fusion beat	Unknown beat
heartbeat types	normal beat (NOR)	atrial premature beat (AP)	premature ventricular contraction (PVC)	fusion of ventricular and normal beat (fVN)	paced beat (P)
	left bundle branch block beat (LBBB)	aberrated atrial premature beat (aAP)	ventricular escape beat (VE)		fusion of paced and normal beat (fPN)
	right bundle branch block beat (RBBB)	nodal (junctional) premature beat (NP)			unclassified beat (U)
	atrial escape beats (AE)	supraventricular premature beat (SP)			
	nodal (junctional) escape beat (NE)				

Figure 2: Beat classes as recommended by AAMI

1.2 Current work

While several methods have been developed to achieve the classification task, the majority of them involve a process of manual feature selection prior to the beat classification process.
We will present a brief outlook of the article by De Chazal and Reilly [5],

which is regarded as a gold standard for classical semi-automated methods of heartbeat classification. The process that the authors describe consists of three steps:

1. preprocessing, which consists on applying certain filters to the signal with the objective of eliminating unwanted noise, as well as beat segmentation and peak detection.

2. manual feature selection, extracting relevant temporal features such as distance between successive beats and morphological features of the P and T waves, as well as of the QRS-complex. These include distances between wave peaks, as well as wave widths and relative position of the different wave peaks, onsets and offsets within the heartbeat.

3. classification of beats,using linear discriminants, into the five AAMI classes.

See Figure 3 for a diagram of the process.

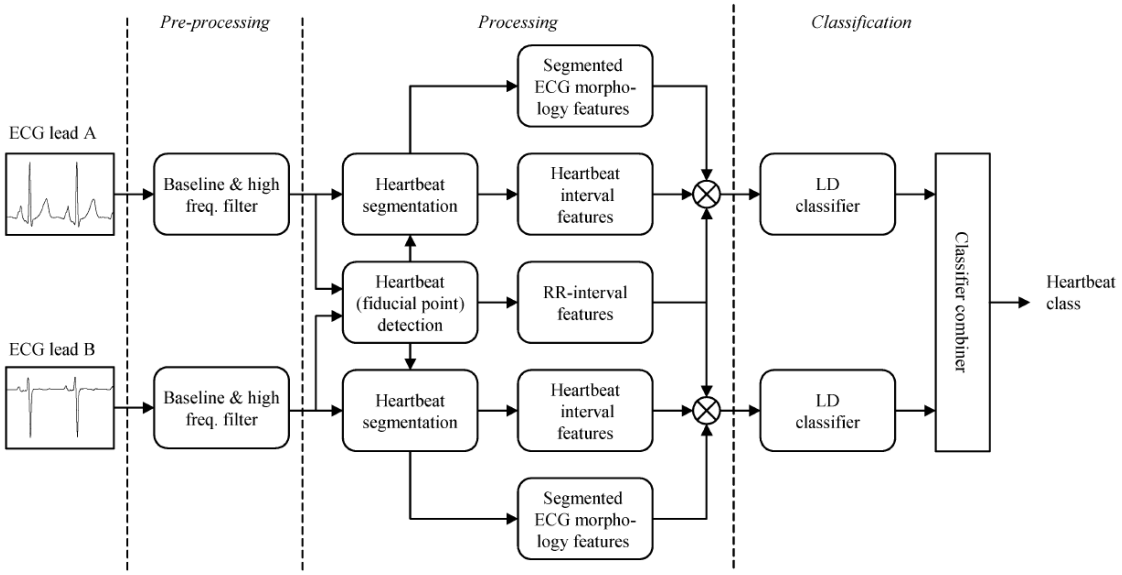


Figure 3: Classifier configuration for the article by de Chazal et al.

Using this method, an overall accuracy of 84.5% was achieved, a specificity of 86.7%, as well as the following sensitivities: S: 53.3%, V: 67.3%, F: 71.6%, Q: 12.5%.

Despite the reasonable efectiveness of the described method, the selection of the features beforehand may not be the best set of features, as it can consist on arbitrary decisions that do not necessarily guarantee an adequate result and can be very time-intensive if done manually due to the large duration of some recordings.
On the other hand, automatic detection of the aforementioned characteristics performed with use of software can produce errors. In light of this fact and with the recent improvements on GPU multi-threading, the deep learning approach has begun to take importance as a viable alternative. Its main objective is to find a suitable feature representation for the raw data in an unsupervised manner and then use this representation to perform a more effective classification. This approach has been implemented in various architechtures such as deep belief networks and convolutional neural networks and has already provided a great performance leap in image and voice recognition tasks.
A reference for this new approach is the paper by Al Rahhal et al. [6], from 2016, on which we will base part of our research. Their approach was the following:
Beats were segmented one-by one and resampled to 50 uniformly distributed points. These served as morphological features. Additionally, 4 temporal features were extracted: pre-RR interval(interval between current beat and previous one), post-RR interval (interval between current beat and next one), local RR-average (10 second average of RR-intervals) and global RR-average (5 minute average).
These 54-dimensional vectors were fed to an autoencoder consisiting of dense layers in order to automatically extract the adequate features in an unsupervised manner. After this training process was completed, a dense softmax layer was appended to the encoder part of the autoencoder to perform multi-class prediction and the network was fine-tuned with backpropagation to classify the beats into their correct classes.
The authors reported an overall accuracy of 97.5%, as well as sensitivities of 37.8% for the S class and 90.1% for the V class.
Later they managed to boost these results by, over several iterations, selecting the beats that generated the most uncertainty in the classification process and asking experts to classify them. We consider that this approach, while effective, reduces the "automatic" character of the process and as such defeats goal that we are aimimg for in terms of time

consumption and ease of use. It is also important to notice that the given performance for the sensitivities was extracted using binary classification, which is a simpler version of the problem and requires separate training for each class.

2 Description of the data set

The MIT-BIH Arrhythmia Database was used because of its availiability, completeness and wide usage in related literature. It contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects studied by the BIH Arrhythmia Laboratory between 1975 and 1979. Twenty-three recordings were chosen at random from a set of 4000 24-hour ambulatory ECG recordings collected from patients at Boston's Beth Israel Hospital; the remaining 25 recordings were selected from the same set to include less common but clinically significant arrhythmias that would not be well-represented in a small random sample.

The recordings were digitized at 360 samples per second per channel with 11-bit resolution over a 10 mV range. Two or more cardiologists independently annotated each record, providing an expert beat classification at each R-wave peak, as well as some other relevant informations about pacing, signal quality, etc.
Below we show a raw sample of a heartbeat belonging to our database.

3 Preprocessing

The recordings were processed with to median filters to remove baseline wander. First a 200 ms filter was applied to extract QRS-complexes and P-waves, followed by a 600-ms filter to extract T-waves. Finally, the resulting signal was processed through a 12-order bandpass filter with cutoff frequency at 35 Hz to remove power-line interference and high-frequency noise.
Next, beat segmentation was performed with the help of *ecg-kit*, a *Matlab* toolbox developed by M. LLamedo Soria [13]. It makes use of the *wavedet* algorithm, a wavelet-based ECG delineator. The software was given the expert's annotations corresponding to the QRS-complex peaks

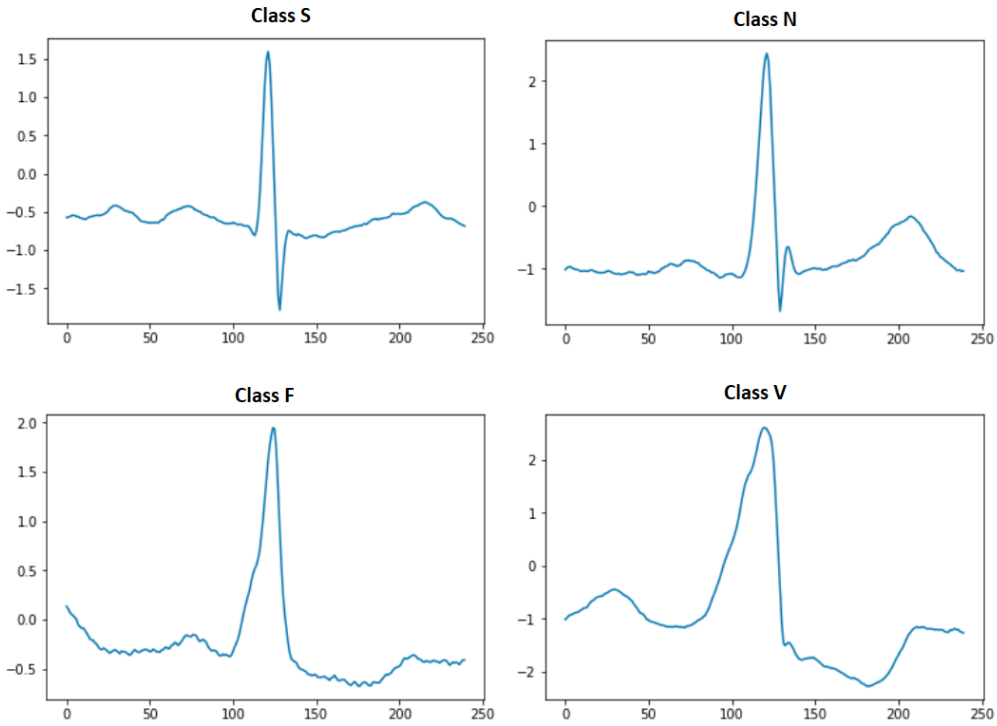


Figure 4: Four beats from MIT-BIH dataset, each representative of a different AAMI class

and generated an additional annotation for the estimated onsets, offsets and peaks of the P,R and T waves. A beat was then defined to start at the P-wave's onset and to end at the T-wave's offset. In the case that a T-wave was not detected, the beginning of the beat was set to be the QRS-wave's onset. An analogous procedure was done in case of a missing T-wave. Finally, if the algorithm failed to provide any of the previous informations, be it due to extreme noise or extreme beat morphology, the beginning of the beat was set to the mid-point between the current beat and the previous one. A parallel procedure was done to determine the beat's end.

These procedures will first be performed during the creation of our baseline classic classifier due to comparability reasons, as the mentioned works [6] and [5] use similar tools, but will later be dropped in the deep learning based classifier because it will be not necessary for our approach.

4 Theoretical introduction

4.1 Machine learning

Machine learning is an area of computer science, more precisely of the field of artificial intelligence. Its main goal is the development of algorithms that give computers the ability to solve complex taks without the need of a definite procedure to do so. On the contrary, these algorithms make use of statistical and optimization tools in order to confer computers with the ability to "learn" in a progressive manner to perform certain tasks. More concisely, a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [17]. Because of the diverse set of tools used in this field, it sits at the crossroad of statistics, mathematics and computer science.

4.1.1 Machine learning tasks

The tasks that are targetted by machine learning algorithms are normally classified into two broad subsets: supervised and unsupervised learning, with finer variants avaliable:

- **Supervised learning:** The computer is presented with example inputs and their desired outputs and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback:
  - **Semi-supervised learning:** the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.
  - **Active learning:** the computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labeling.
  - **Reinforcement learning:** training data (in form of rewards and punishments) is given only as feedback to the program's

actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

- **Unsupervised learning:** No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

4.1.2 Dataset partitioning

The complete set of exaples at the computer's disposal is called the dataset. For learning purposes it is usually separated into three subsets:

- **Training set:** During training, the computer has access to the information in this set and tries to devise an adequate representation of it, finding patterns.
- **Testing set:** During each training iteration, the performance of the current representation will be checked against the examples in the training set and the algorithm will act trying to maximize the performance on this set.
- **Validation set:** After the training process is completed, it is useful to check the performance of the obtained model against an additional set of data points so as to evaluate the model's capacity to generalize well to new information.

4.1.3 Optimization in machine learning

In order to minimize the error of machine learning algorithms, iterative methods are used. The most common of them is stochastic gradient descent, for which several, more refined variants have been recently created. In machine learning, large training sets are necessary for good generalization ability. However, large training sets are also more computationally expensive.

The cost function used by a machine learning algorithm often decomposes as a sum over training examples of some loss function:

Suppose that the model can be optimized according to some parameter θ, then this loss function can be expressed as

$$Q(\theta, \mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} Q_i(\theta, x^i, y^i) \tag{1}$$

where $\mathbf{x} = \{x^i\}_{i=1,\dots,n}$ are the observations and $\mathbf{y} = \{y^i\}_{i=1,\dots,n}$ their true classes (in one were dealing with an unsupervised classification problem, Q would only depend on the observations). The parameter ω which minimizes $Q(w)$ is to be determined. Note that each Q_i depends only on the i-th element in the data set. Examples of such functions are mean squared error or the negative log-likelyhood of the parameter, conditioned to the observations. For such additive cost functions, the classical algorithm of gradient descent relies of computing

$$\nabla_\theta Q(\theta, \mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta Q_i(\theta, x^i, y^i) \tag{2}$$

and updating the parameter θ by moving a certain amount along the direction of the negative gradient (or positive if one considers maximization). The issue with this approach is that the computational cost of a single iteration grows as $O(n)$, implying prohibitive calculation times for large amounts of data.
Stochastic gradient descent solves this by selecting a subsample of the data set, usally called a minibatch, with size $m < n$ typically ranging from 10 to a few hundreds, and aproximating the whole gradient by

$$\tilde{\nabla}_\theta Q(\theta, \mathbf{x}, \mathbf{y}) = \frac{1}{m} \sum_{k=1}^{m} \nabla_\theta Q_{i_k}(\theta, x^{i_k}, y^{i_k}) \tag{3}$$

and making the following parameter update at each iteration:

$$\theta \to \theta - \epsilon \nabla Q(\theta), \tag{4}$$

where ϵ is a model-specific parameter called the learning rate.
Additionally, a *momentum* term can be added to the uptate to improve speed:
Stochastic gradient descent with momentum remembers the update $\delta\omega$ at each iteration, and determines the next update as a linear combination of the gradient and the previous update. This can help reduce big

oscillations in the update direction, as the previous motion tends to be conserved: previously, the size of the step was simply the norm of the gradient multiplied by the learning rate. Now, the size of the step depends on how large and how aligned a sequence of gradients are
Clearly this method draws inspiration from the case of a particle moving through a slope, which has a certain momentum due to its velocity. Here $\delta\omega$ can be interpreted as the particle's momentum and its coefficient in the update as a friction.

4.1.4 Advantages and disadvantages of the machine learning approach

The classic approach to solving a problem consists in analyzing its characteristics, understanding it and finding a deterministic and task-specific procedure to do so. The problem with this approach lies in the fact that some poblems can be extremely hard, even impossible to be described in clear mathematical terms. Even if they theoretically could, most of the times is is not clear whether there exists a "good" way to solve them.
Machine learning algorithms solve this problem by considering existing data relevant to the proposed task and trying to find patterns in their structure. This approach finds its inspiration in the way that our brain opperates when faced with a novel task: by experience and a process of trial and error.
While this way of solving things can be error-prone and tedious, once the learning process is successfully completed, new related tasks can be performed in nearly linear time with respect to the size of entry data. Therefore, machine learning algorithms are particularly well suited for real-time applications.
While these are appealing advantages, there also exist some caveats with this approach:
First off, for this strategy to work, large amounts of data have to be avaliable in order to have sufficient material for a successful training phase. Additionally, there is still a lot of research to be done regarding the interpretation of the algorithm's results, as for now they are treated as a sort of "black box" due to the large number of parameters involved and the complex internal structure of these programs.

4.2 Deep Learning

Simple machine learning algorithms like k-means clustering or principal component analysis work well on a wide variety of important problems. However, they have not been successful in solving the central problems in AI, such as speech or object recognition, amongst others. The development of deep learning was motivated in part by the failure of traditional algorithms to generalize well on such AI tasks.
The challenge of generalizing to new examples becomes exponentially more difficult when working with high-dimensional data, and the mechanisms used to achieve generalization in traditional machine learning are insufficient to learn complicated functions in high-dimensional spaces. Such spaces also often impose high computational costs. Deep learning was designed to overcome these and other obstacles.

4.2.1 Deep feedforward networks

Deep feedforward networks, also called feedforward neural networks, multilayer perceptrons, or simply neural networks, are the main common structure of the majority of deep learning models. The goal of a feedforward network is to approximate some function f. For example,a classier associates a class y to an input x through said function f. A feedforward network denes this task as $y = f(x;\theta)$, where θ are some set of parameters and learns the values of the parameters that yield the best approximation to the true results. They are called *deep* networks because the function f is obtained via a composition of different $f_1, ..., f_n$ and normally represented as a directed acyclic graph, where each edge represents one of these functions.

We speak of neural networks because an analogy can be established between them and brain function: As the entry to feedforward networks is vector-valued, each of its dimensions can be intepreted as a neuron. Instead of seeing the layer as representing one vector-to-vector function, we can also think of the it as a set of units that act simultaneously, each representing a vector-to-scalar function. Each unit resembles a neuron, as it receives inputs form its surrounding units and computes an output that can in turn be passed to other neurons.
The fact that several functions (layers) are used is also inspired by the

functioning of human reasoning, as it is believed that neurons act in a sequential manner, forming different levels of abstraction.

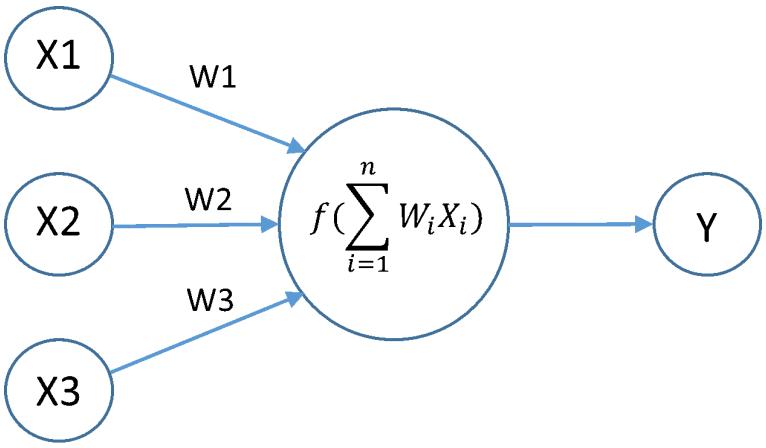


Figure 5: Visualization of a neural network

4.2.2 Activation functions

Suppose that we have an element $u^i = (u^i_j)_j$ belonging to the i-th layer of a neural network. The function f_i that maps this element from layer i to layer $i+1$ will be expressed in the following way:

$$f_i(u^i) = g(\sum_{j=0}^{n} \omega^i_j u^i_j),$$

where g is some non-linear function and the ω^i_j are a set of weights for the i-th layer that need to be optimized. We call this function g an activation function and it is chosen to be a non-linear function with the aim of being able to represent more complex relationships between the data that a merely linear model would not be capable of finding.
There are several functions commonly used as activation functions for neural networks:

- Sigmoid:

$$g(x) = \frac{1}{1+e^{-x}}$$

Its advantages are its smoothness, as well as having a bounded range between zero and one. On the other hand, for values sufficiently par appart from zero, its gradient becomes small. This can lead to a very slow, even not converging model.

- Rectified Linear Unit (ReLU):

$$g(x) = max(0, x)$$

It has two main advantages: for values greater than zero the gradient is constant, thus speeding up computations, which leads to faster convergence when compared with the use of sigmoid activations. Additionally, it tends to yield sparse representations of data points, as negative values vanish. This is an advantage, as deep learning models deal with large amounts of data , which can be hard to store. A sparse representation can help achieve efficiency in this regard.
It has, however, the disadvantage of a non-bounded range, meaning that certain values can experience a blow-up during calculations.

- Softmax:

$$g_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

It is normally used in the final layer of multi-class classification models, as its outputs are in the range $(0, 1)$ and they sum to 1. Therefore they can be interpreted as probabilities for each class. The purpose of using the exponential function is to accentuate the differences between the different computed values, thus being near to an indicator function but with differentiability.

In later times, ReLU is being increasingly favoured over sigmoid mainly for its faster performance times [15].

4.2.3 Loss functions

Loss functions serve as a metric by which to determine if the designed model architecture is effective in solving the problem in hand. In this work we will use the following two:

- **Mean squared error**: Let $\bar{x}$ be an approximation of x:

$$\sum_{i=1}^{n} ||x_i - \bar{x}_i||^2$$

  It is favoured because its mathematical convenience, as it is a convex function and it represents the variance in case of an unbiased estimator, but has the disadvantage of heavily weighting outliers.

- **Categorical cross-entropy**: Let p_i be the real probability of class i and $\bar{p}_i$ be the probability of class i under the model's predictions. Then the categorical cross-entropy function can be expressed as

$$\sum_{i=1}^{n} p_i \log\left(\frac{1}{\bar{p}_i}\right)$$

4.2.4 Dropout

We say that a model's layer has a dropout rate $p \in (0, 1)$ if in each iteration of the optimization process each neuron has a probability p of being activated. This procedure forces the deep learning algorithm make more robust predictions, as each neuron will be more independent of the others. As an effect, the model's generalization ability can be boosted, reducing overfitting.

4.2.5 Auto-encoders

The auto-encoder [11] is a deep learning model that aims to learn a representation of a data set in a different space in a way that the error of reconstructing the data from this representation space back to the original is as small as possible. The model will be learning the identity function.

Its structure consists of two symmetrical encoding and decoding parts, each possibly having multiple hidden layers. The middle layer will be called the representation layer.

Auto-encoders are used mainly for two resons: Firstly they can be useful for compressing data high-dimensional data into considerably smaller spaces. Moreover, they are used as a method for finding alternative representations of data, new features, that are able to better characterize it for further operations such as classification.

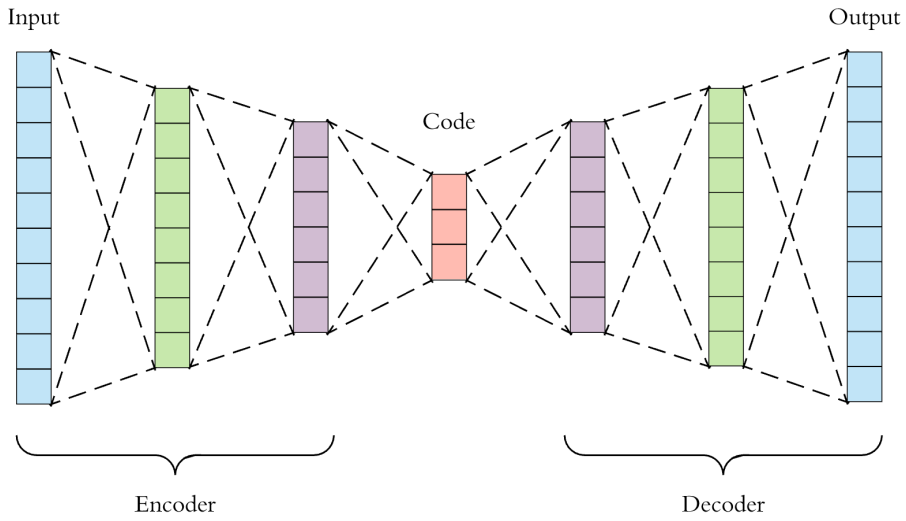


Figure 6: Structure of an auto-encoder

4.3 Convolutional neural networks

Convolutional neural networks [14] (CNNs), are a specialized kind of neural network for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, or images, which can be thought of as a 2-D grid of pixels.

4.3.1 Convolution

Traditional neural network layers use matrix multiplication by a matrix of parameters, the weights, with a separate parameter describing the interaction between each input unit and each output unit. This means that every output unit interacts with every input unit. Convolutional networks, however, typically have sparse interactions: each neuron in a layer is only connected to the k nearest neurons in the next layer. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations. These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires $m * n$ parameters, and the algorithms used in practice have $O(m * n)$ runtime. If we limit

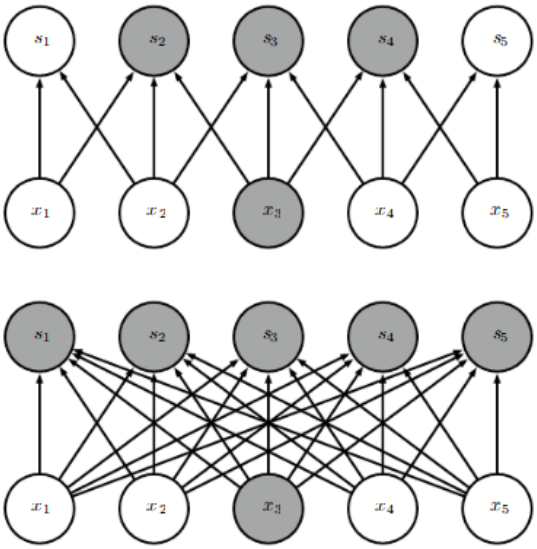


Figure 7: Comparison between conecticivity of a dense layer (below) and a convolutional layer(above)

the number of connections each output may have to k, then the sparsely connected approach requires only $k * n$ parameters and $O(k * n)$ runtime.

The property of sparse conectivity is obtained via a discrete convolution of the input layer with a kernel of size k. Let $\mathbf{x}$ be the input of a layer, then its output $\mathbf{s}$ is computed in the following way:

$$s_i = \sum_{j=0}^{k} x_j K(i-j)$$

where K is a function that can be learned and is called the kernel. The generated $\mathbf{s}$ is called the feature map. We therefore only need to learn K in order to know the output of the layer.
In general, a certain number of different kernels are learnt with the purpose of each of them each learning a different representation.
Each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels). The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. This reduces storage requirements of the model compared to dense neural networks.

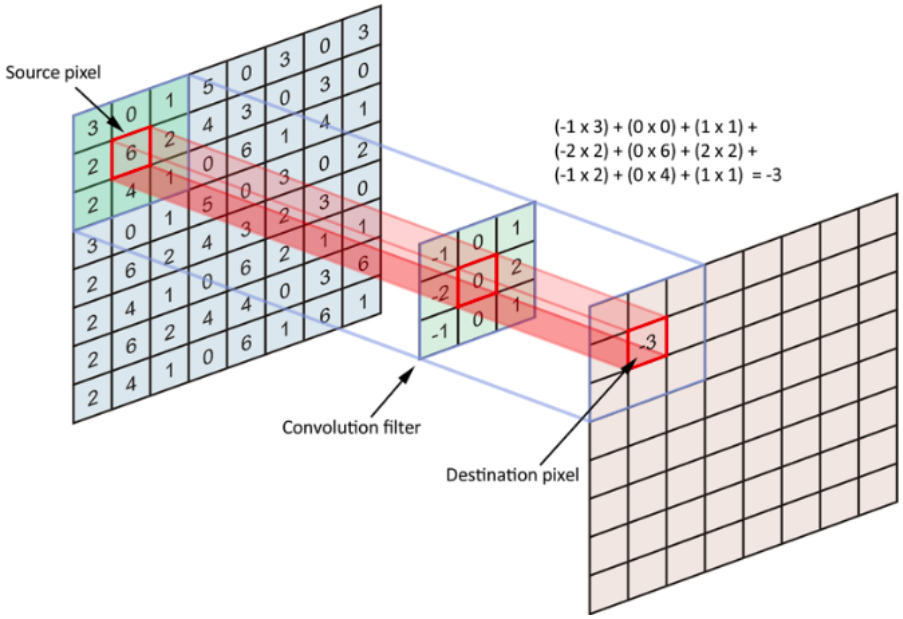


Figure 8: Visualization of the convolution process for an image. Note that we are dealing with a time series, which is one-dimensional.

4.3.2 Pooling

Pooling is a frequent addition to convolution. It consists on replacing the output of a layer in a certain region with its maximum, average, L^2 norm, or other statistics. It serves two main purposes: it reduces the dimensionality of representations and it helps to make the representation approximately invariant to small translations of the input, helping the model to become more resilient to noise.
The most common forms of pooling are max-pooling, which consists on computing the maximum of outputs in a region, and average pooling, which in turn computes their average.

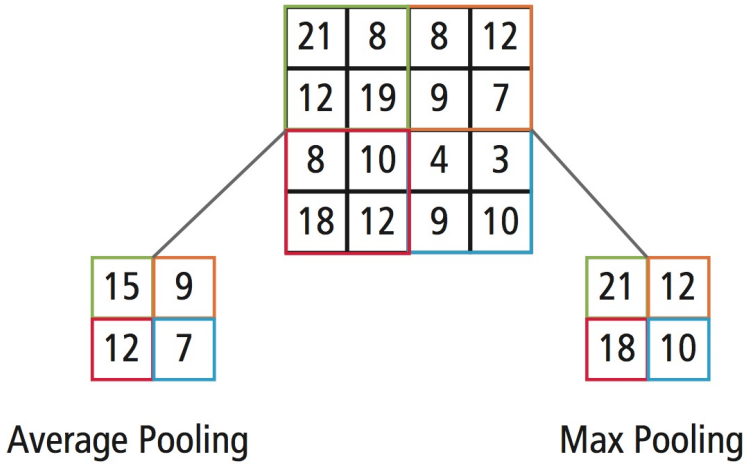


Figure 9: Visualization of the pooling process for a 2-D image

5 Classical classifier

5.1 Feature extraction

In this section we will perform a process of feature extraction that is inspired in the approach by [6]. We consider that it contains the essential information in a heartbeat, while being simple enough. Each beat was represented as a vector containing 54 elements:
The first and second characteristics were chosen to be the pre-RR and post-RR intervals, which indicate the time elapsed between the current beat's fiducial point and the previous (resp. next) one.
For the third caracteristic an average of the 10 previous RR-intervals was chosen so as to represent the local dynamic of the beats
For the fourth characteristic a broader average of RR-intervals was chosen, comprising the past 5 minutes and capturing a more patient-specific property.
The remaining points of information were dedicated to beat morphology: a cubic spline interpolation was performed in order to extract 50 evenly spaced points.
This procedure finally resulted in a 83702 by 54 matrix.

5.2 Beat classification

Class Q was discarded from classification due to it having only 15 representatives. Furthermore it corresponds to beats that the cardiologists were not able to classify and therefore has no medical relevance.
Several classifier architectures were tested, including decision trees, linear discriminant analysis and k-nearest neighbours, the latter being the best-performing one.

To keep track of model the following performance measures will be used: overall performance (OP), sensitivity (Se), positive predictive value (P+) and area under the ROC curve (AUC). The S, F and V classes will each separately be tested against the remaining classes to evaluate these parameters.
We split the dataset in two subsets, one for training, contining records 101, 106, 108, 109, 112, 114, 115, 116, 118, 119, 122, 124, 201, 203, 205, 207, 208, 209, 215, 220, 223 and 230, with the remaining records for validation.

This choice is made because the same partition is performed in the works by Rahhal et al [6] and de Chazal et al [5], therefore increasing comparability of the results.
Linear discriminant analysis offered an accuracy of 93% but particularly poor sensitivity values regarding F and S classes, respectively 15% and 25%. For this reason it was not deemed adequate, as the ability to detect the anomalous beats with sufficient frequency is prioritary (Table 1).
The best overall accuracy that could be attained with decision trees was 96.5%, with a limit of 100 splits. The model attained acceptable positive pretictivity ranging from 58% for de F class up to 97% for the N class. Nevertheless, a sub-par sensitivity for the F class of 25% harmed the model (Table 1).

The best-performing model was k nearest neighbours and several configurations were explored. For distance weighting, inverse distance, inverse squared or none at all; for number of neighbours: 1, 3, 10 and 100. It was observed that for this problem an increase of the number of neighbours above 3 was not conductive to greater accuracy, in fact for values of k greater than 10 performance was diminished. As for the distance weighting, the inverse of the distance squared was the optimal choice (see Table 2). That meant that neighbours not sufficiently near were penalized by the algorithm.
The ROC curves presented below show the relation between false positive rates and true positive rates for the respective classes in k nearest neighbours classification when the discrimination threshold is varied. We selected the threshold that offered the best ratio. Furthermore, the area under this curve gives us a measure of perfromance of the chosen model. In the following figures, said curves and a confusion matrix will be presented for the optimal k-nearest neighbours model in terms of sensitivity, which uses 1 neighbour, as well as some further performance measures comparing all the methods tried.

positive class		S			V			F		
measures	OP	Se	P+	AUC	Se	P+	AUC	Se	P+	AUC
l. discriminant	.93	.25	.56	.91	.56	.73	.94	.15	.19	.86
decision tree	.69	.96.5	.83	.95	.79	.90	.95	.25	.58	.86
1 neighbour	.982	.81	.90	.90	.94	.95	.97	.67	.73	.83
3 neighbours	.982	.78	.93	.93	.93	.96	.98	.60	.79	.89
10 neighbours	.980	.73	.96	.95	.91	.96	.99	.53	.81	.91
100 neighbours	.965	.57	.98	.96	.79	.96	.99	.20	.85	.95

Table 1: Comparison of performance between the different classification schemes

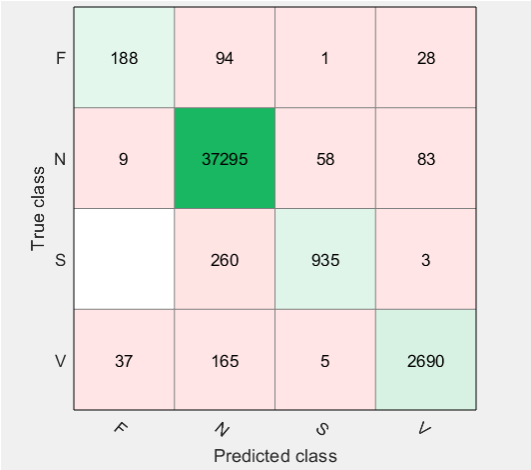
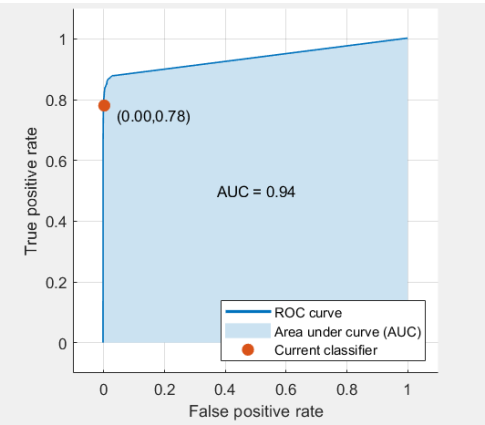
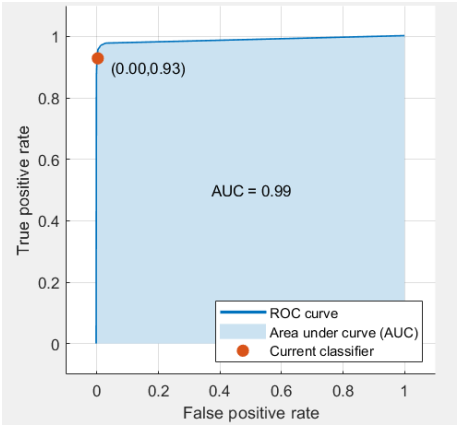


Figure 10: Confusion matrix for KNN using 1 neighbour



(a) S class vs. the other classes



(b) V class vs. the other classes

Figure 11: ROC curves for KNN using 1 neighbour

5.3 Deep learning based classifier

5.3.1 Motivation

The idea of designing a heartbeat classifier that is based on deep learning follows two main motivations:
We want to develop a tool that is able to detect with sufficient prediction the presence of anomalous beats in real-time. As such, the use of such a model can be very advantageous, as hard computational work will be done in the training phase, for which we have large amounts of data at our disposal.
We also have the intention of not performing any pre-processing to the input data, other than partitioning the data set into the several beats, which would further speed up the classification process.

5.3.2 General structure of the proposed models

All the models that we will propose will consist of two main structures:
In the first stage we will implement an auto-encoder. The reasoning behind it is that we want to find alternative ways to represent the ECG data without having to resort to possibly arbitrary considerations such as R-R intervals. We expect the auto-encoder to find the most suitable representation for a future classification task.
Furthermore, the auto-encoder will perform dimensionality reduction, which can be useful for storage of large amounts of data.

On top of the auto-encoder layer and after it is trained to reconstruct the data correctly, we will append an additional layer for classification. It will consist of four output neurons, each associated to one of the four unique classes of beat types. For each example, the neuron with the highest activation will be interpreted as the predicted class.
The whole model will be later fine-tuned in order to classify the heartbeats of a test dataset.

The model will be created using *Keras* [21] with *Tensorflow* [22] backend. The hardware used is a *Nvidia GTX 1080* GPU.

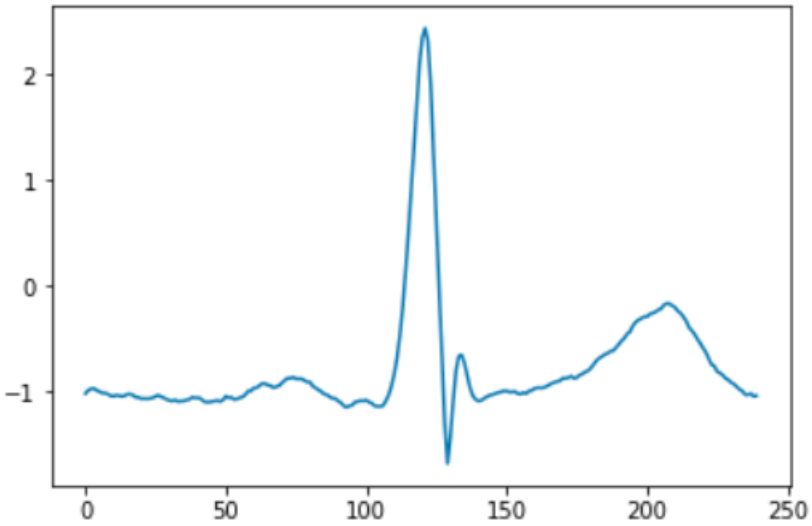


Figure 12: Example of a training example, the beat contains 240 samples

5.3.3 Data pre-processing

Each training example will consist of one heartbeat. We sample them, locating the center of the heartbeat in the center of the sample. Note that the sampling frequency is 360Hz, therefore we have determined through a heuristic process that each beat will contain 240 samples. We believe that the exact size of the interval is not of great importnce, as this width is sufficient to capture the QRS-complex and de P and T waves, which are the most salient features of the ECG.
Figure 12 is an example of a beat sample as is used for classification.

5.3.4 The auto-encoder

First proposal: dense autoencoder We will implement an auto-encoder consisting of two hidden layers. As the input dimension is 240, we choose the first hidden layer to have 120 neurons and we leave the dimension of the representation layer as a hyperparameter of the model to be fine-tuned. We consider only one such layer, as the benefit of possible additional layers does not in general achieve much greater performance, but would drastically increase the number of model's parameters and thus training time [15].

We optimize by calculating the mean squared error between the inputs and their reconstuctions. In addition we will introduce a Kullback Leibler divergence term in the loss function to force the auto-encoder to yield sparse representations of the data. This, we believe, will contribute to the activation of only a small ammount of neurons for each training example, thus possibly forcing them to activate similarly when beats correspond to the equal classes. The loss function will thus have the form

$$L(\mathbf{x};\theta) = \sum_{i=1}^{n} ||x_i - \bar{x}_i||^{+} L^{'},$$

where the term $L^{'}$ represents the cross-entropy between the mean activation function of the hidden layers and a target mean activation set to a low value (0.001 in our case). This addition is to ensure a sparse representation.
We will train all the models using the *Adam* optimizer, which is a modification of stochastic gradient descent offering faster convergence and better stability in the majority of applications [16]. For this structure of an autoencoder, we utilize the recommended learning rate of 0.001 [16]. As activation function, we will first select ReLU, but later we will consider also Sigmoid and compare performances.

As we desire a marked dimensionality reduction, we will explore the following sizes of representation layers: 8,12,16 and 20. They all represent a more than 10-fold reduction over the original dimesion and yield a representation layer with a number of neuron between 2 and 5 times the number of heartbeat classes.
We will choose size 12 for the representation layer, as it strikes a balance between performance and compactness: 16 layers do not achive noticeably greater performance, while 20 improves on it, but almost doubles the representation dimensionality (see Figure 13).

Next we will check if the change of activation function to sigmoid would bring about a performance increase. First, it is important to note that the final layer of the decoder will not be assigned an activation function, as these two are positive valued, but our heartbeats can have a negative range.
Clearly ReLU activation performs better in this problem, and for this

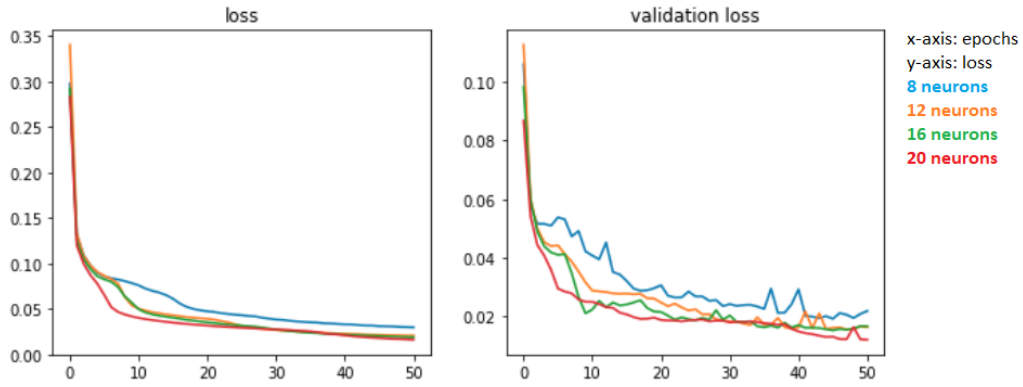


Figure 13: Training results for different sizes of the representation layer

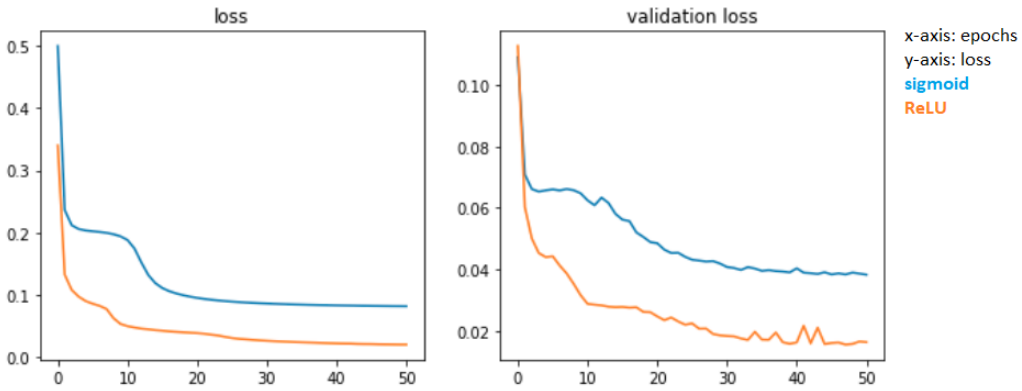


Figure 14: Training results for different activations

reason we will keep it over sigmoid (see Figure 14).

The next proposed modification is the addition of dropout to both hidden encoding layers as well as input layer. We believe that this will help reduce overfitting, and help generalize to other unseen beats. Following recommendations from [8], we will ad a dropout rate of 0.8 to the input layer and 0.5 to all hidden layers. Additionally, we will also follow suggestions from [8] increasing ten times the learning rate and doubling the size of the representation layer. We see that training performance is boosted with the addition of dropout. The average reconstruction error over the test set is 0.0162 (See Figure 15).

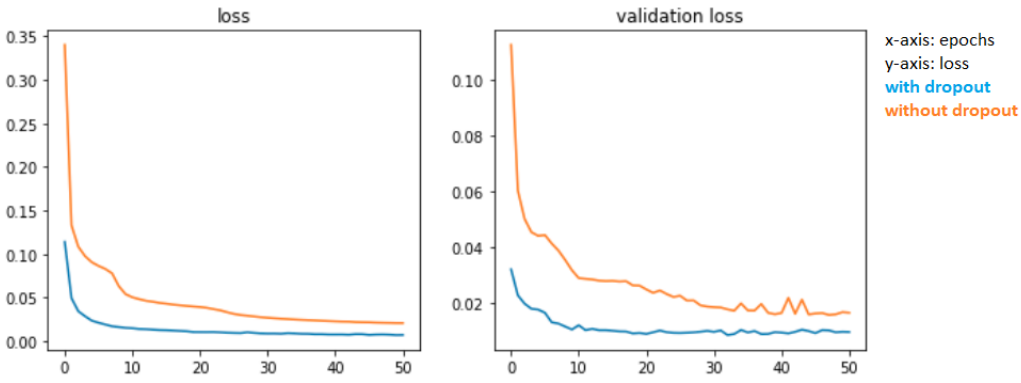


Figure 15: Changes in training results for the addition of dropout

Second proposal: convolutional autoencoder We propose an upgrade of the previous model by adding convolutional layers. We believe that this change is justified due to several reasons:
Previously we have only considered the main ECG lead. The addition of a convolutional first layer enables us to process multi-dimensional time series, thus being able to feed to the model the two availiable leads at the same time. This can be helpful because some beats that may apear very similar in the main lead can exhibit different structures viewed in the other lead, as it captures electrical impulses from another angle. Moreover, ECG data tends to be very noisy, with small oscillations that do not contribute to the classification of the beats. By adding a convolutional filter we are forcing nearby neurons (points) to have similar activations, thus neglecting the finer, unnecessary, structure of the data. To achieve this, we propose adding a convolutional layer of size 30. This corresponds in time to a typical width of the QRS-complex, P and T waves. By first convolving with a filter of this size, the algorithm will allegedly detect the forms and positions of these main features. On top of this layer the previouly devised auto-encoder will be appended.
The model was then trained for 50 epochs with a learning rate of 0.0001.

Its mean squared reconstruction error was evaluated to be 0.0015, which is over 10 times superior to the dense auto-encoder's performance.

In the following figure one can appreciate the reconstruction ability of

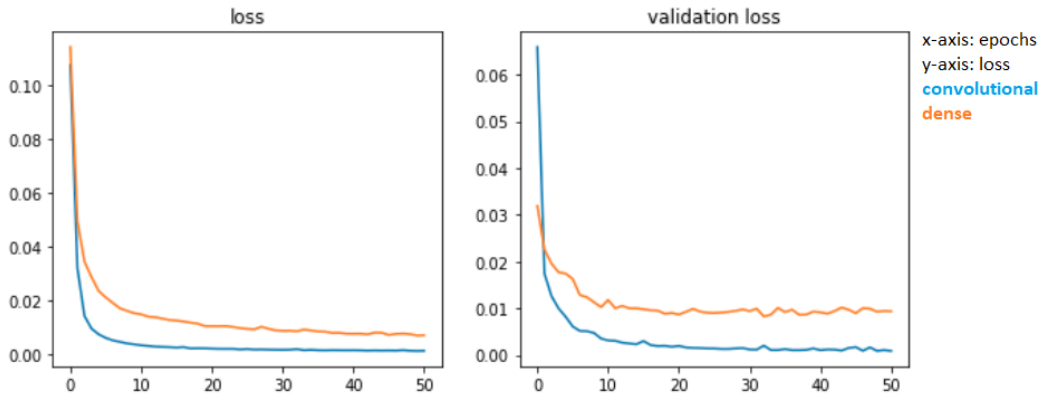


Figure 16: Comparison between dense and convolutional auto-encoder

the convolutional auto-encoder: The main features such as P and T waves and the QRS-complex are precisely reproduced. We want to remark that the input of the auto-encoder consists of two 240-dimensional vectors (one for each lead), and that the found representation of dimension 24 is able to maintain a high level of information about the original output so as to yield the displayed reconstruction. We believe that our auto-

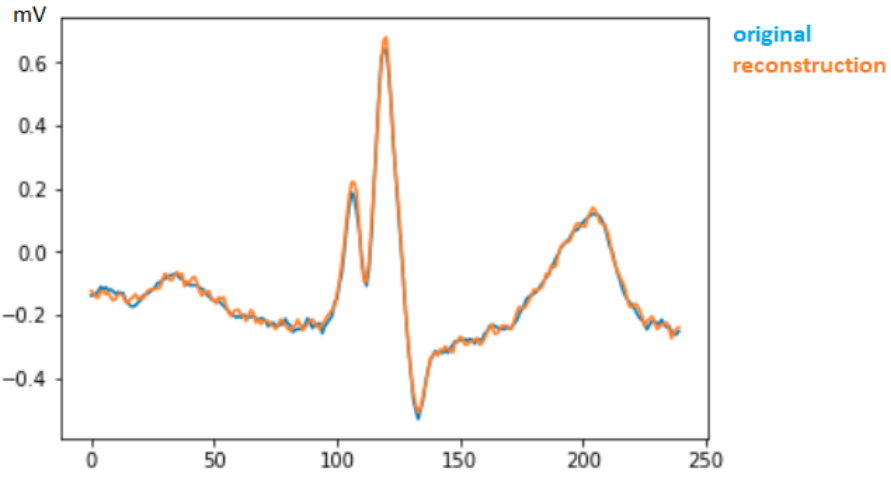


Figure 17: Reconstruction of a beat by the convolutional auto-encoder

encoder can serve as a useful tool for medical centers which have collect large amounts of ECG data on a daily basis, as this method would be able to store the records with ten times less memory usage while having very small loss of information.

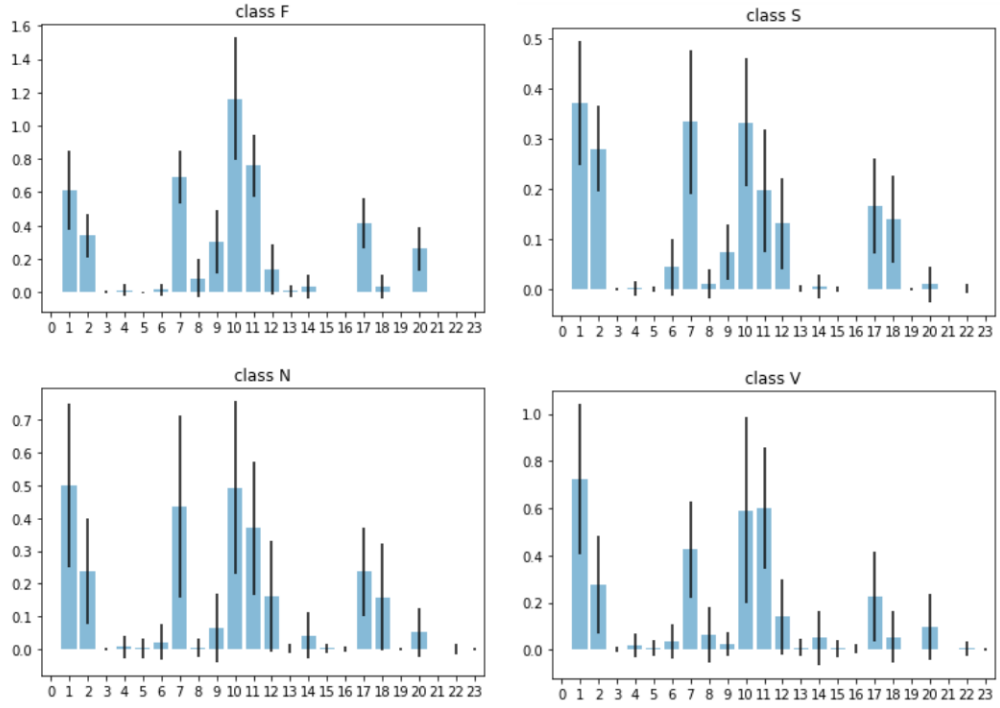


Figure 18: The bars represent mean activations of the neurons over all training exmples for the 4 AAMI classes, while the black lines depict their variance.

5.3.5 Classification

Our proposal is to append to the previously learned encoding an additional dense layer using a softmax activation. It will have four output neurons and will be the final layer.
After the auto-encoder is trained, we extract the learned weights for the encoding part and use them to initialize the network. Then the whole model is trained to predict the correct labels in the training dataset. To evaluate the training results we will use the categorical cross-entropy loss function. We decided to apply a weight penalty to each class depending on its relative frequency, as the data set is very unbalance, with more than 90% of examples belonging to class N. As such, misclassifications of the remaining three types of beats were given greater contribution to the training loss. If this procedure is not done, the classifier simply assigns all the predictions to the N-class, as doing this achieves a very high overall accuracy.

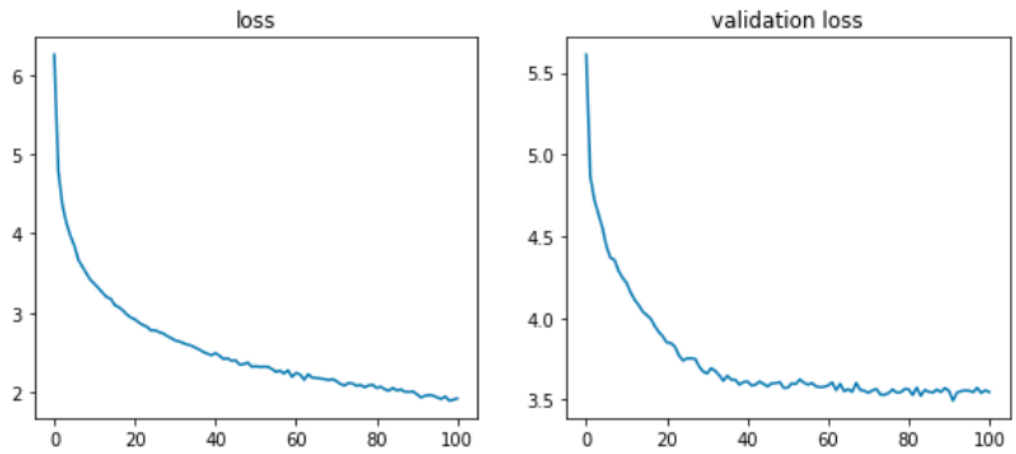


Figure 19: Training loss for classification

We will present the results in terms of sensitivity with respect to the different classes, as this is a medical problem that requires a high ability to detect anomalous beats, as they can be indicative of severe medical issues:

We obtained sensitivities of 90.2% for the S class, 56.4% for the N-class and 84.4% for the V-class.
However, class F was classified with a sensitivity of 3.7%. In the majority of cases, these beats assigned to the N-class. We believe that the cause of the faulty classification of F beats is due to this model's inability to capture the beat's rythmic patterns, as some of the beats need to be analyzed in conjunction with its surrounding beats in order to detect arrhythmias.

Third proposal: extending the range of input data To remedy the previous issue, we consider modifying the sampling width of the dataset: instead of taking one beat as one training example, we will consider a training example be a four second window centered around a beat. The choice of precisely four seconds was made after analyzing the performance of window sizes ranging from two to ten seconds. It is

Model	S	N	V	F
1 beat	0.902	0.564	0.037	0.844
4 seconds	0.800	0.693	0.581	0.859

Table 2: Sensitivities comparison between current model (4 seconds) and previous model (1 beat)

sufficiently large to capture the two beats surrounding the central beat, and small enough to still keep the problem's dimensionality in reasonable margins.
To exploit the advantages of this modification, add an additional filter to the convolutional part of the autoencoder: it has a width of 360, which corresponds to one second. The reasoning behind this addition is that this filter will capture information related to heart rythm, therefore yielding varying response when heart rate is altered with respect to normal parameters.

The new obtained sensitivities are the following:
80% for the S-class, 69.3% for the N-class, 85.9% for the V-class and 58.1% for the F-class.
Note the vast improvement in classification of this last class, compared with the previous approach, while maintaining similar or better sensitivities for the remaining classes.

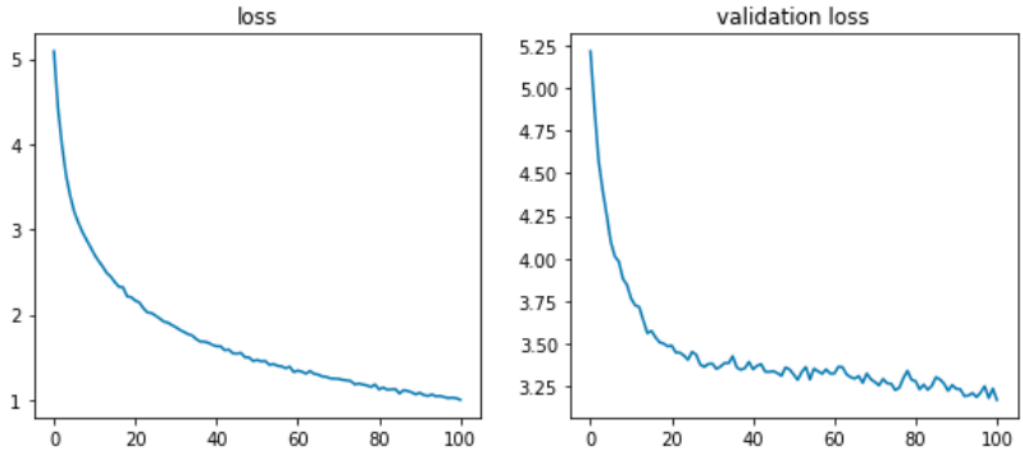


Figure 20: Training loss considering 4-second windows around beats

Model	S	V	F
1-nearest neighbour	0.81	0.94	0.67
Al-Rahhal et al.	0.378	0.901	—
De Chazal et al.	0.573	0.673	0.716
our model	0.800	0.581	0.859

Table 3: Comparison of class sensitivities across the analyzed models.

6 Conclusions

We have achieved a classification framework that offers comparable class sensitivities to the discussed works by De Chazal et al. [5] and Rahhal et al. [6], as well as the previously developed k-nearest neighbours classification, as shown in Table 3, while keeping preprocessing needs to a minimum: we do not choose to extract temporal ECG features through use of auxiliary software, nor do we filter the input data. This will allow more efficient real-time prediction taks.

Notice that while our model has the best performance in regards to the detection of heartbeats in classes S and F, it performs rather worse when detecting beats in class V. We conjecture that this is due to the design choices that we have made for the autoencoder. Therefore we see a potential for improvement in this area, with the goal of finding another-possibly convolutional-layer, which is able to capture a fundamental property of the V beats. This would possibly require a greater knowledge of the biological principles underlying the human heart's functioning.
It is also interesting to observe that the nearest neighbours algorithm performs similarly to the models based on deep learning in terms of sensitivity, while being much more computationally efficient. This raises the question whether deep learning is the optimal approach to the problem of heartbeat classification if one has sufficiently well filtered and preprocessed data, or if simply there exists a better neural architecture that has not yet been found.

On another note, we believe that the dimensionality reduction performed by the convolutional auto-encoder, which achieves c.a. 20-fold compression, can be helpful as a means of data reduction for efficient

storage of large amounts of clinical data in health centres.

In futher work we would like to explore the use of LSTM (long short term memory) [18] to encode beats, as they have been proven successful for the analysis of time-dependent data [19]. Furthermore, we would want to analyze the use of the newly devised (2017) Scaled Exponential Linear Unit (SELU) [20] as activation for our auto-encoder. It has been proven to offer an upper and lower bound on the variance of neuron activations, thus eliminating the common problems of vanishing and exploding gradients and allowing for deeper architectures that could potentially offer more abstract representations.

References

[1] WHO fact sheet.
`http://www.who.int/mediacentre/factsheets/fs310/en`

[2] D. Jiwoong Im, S. Ahn, R. Memisevic, Y. Bengio.
Denoising criterion for variational autoencoding framework. ArXiv, November 2016

[3] B. Benitoa, J. Brugada, R. Brugada, P. Brugada.
Síndrome de Brugada. Rev Esp Cardiol. 2009; 62(11):1297-315

[4] *testing and reporting results of cardiac rythm and ST segment cardiac algorithms*. Association for the Advancement of Medical Instrumentation, 1998.

[5] P. de Chazal, M. O'Dwyer, R. B. Reilly.
Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features. IEEE transactions on biomedical engineering, vol. 51, no. 7, July 2004.

[6] M.M. Al Rahhal, Yakoub Bazi, Haikel AlHichri, Naif Alajlan, Farid Melgani, R.R. Yager.
Deep learning approach for active classification of electrocardiogram signals. ArXiv, 2016

[7] K. Sohn, X. Yan, H. Lee.
Learning Structured Output Representation using Deep Conditional Generative Models. Advances in Neural Information Processing Systems 28, 2015.

[8] N.Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov.
Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer.
SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research archive,Volume 16 Issue 1, January 2002, 321-357

[10] Svetlana Lazebnik.
CS 598 LAZ: Cutting-Edge Trends in Deep Learning and Recognition. University of Illinois, 2017.

[11] Diederik P Kingma, Max Welling.
Auto-Encoding Variational Bayes. ArXiv, 2013

[12] Trevor Hastie, Robert Tibshirani, Jerome Friedman
The Elements of Statistical Learning: Data Mining, Inference, and Prediction.

[13] M. Llamedo Soria
ecgkit
`http://marianux.github.io/ecg-kit`

[14] Y. LeCun, Y. Bengio
Convolutional Networks for Images, Speech and Time Series

[15] I. Goodfellow, Y. Bengio, A. Courville
Deep Learning. MIT University Press, 2015

[16] D. P. Kingma, J. Ba
Adam: A Method for Stochastic Optimization. 3rd International Conference for Learning Representations, San Diego, 2015

[17] T. Mitchell
Machine Learning. Mc-Graw Hill, 1997

[18] S. Hochreiter, J. Schmidhuber
Long short-term memory. Neural Computation 9(8):1735-1780, 1997

[19] P. Malhotra, L. Vig, G. Shroff, P. Agarwal
Long Short Term Memory Networks for Anomaly Detection in Time Series. ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015

[20] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter
Self-Normalizing Neural Networks. Advances in Neural Information Processing Systems 30 (NIPS 2017)

[21] F. Chollet
Keras
`https://keras.io`

[22] Google Brain
Tensorflow
`https://www.tensorflow.org`